Utilizing Unused Resources To Improve Checkpoint Performance

Ross Miller & Scott Atchley

Oak Ridge National Laboratory
Leadership Computing Facility

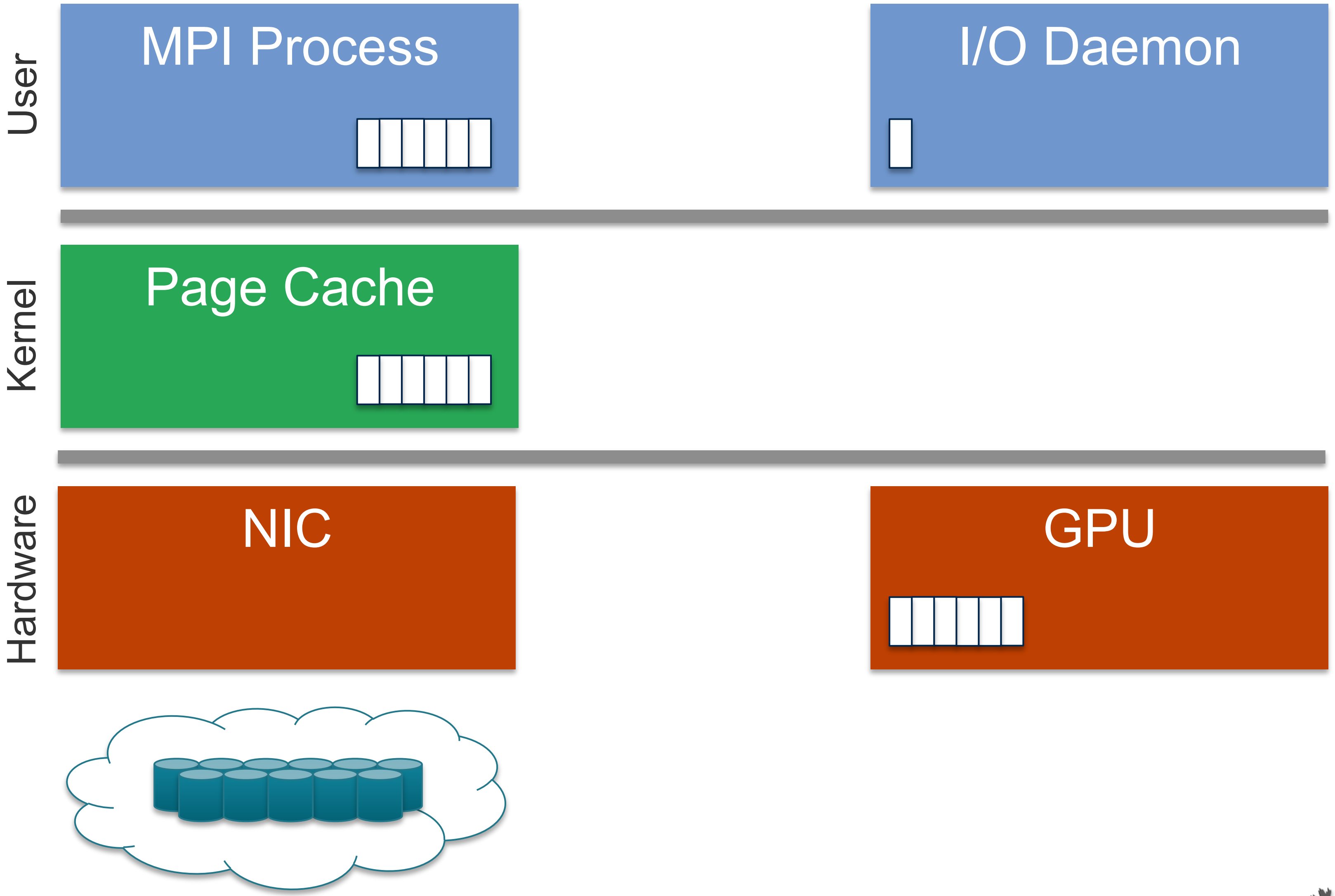
Filesystem Performance

- Users want faster I/O
- Less performance variability would be nice, too
- Mostly write performance (not a lot of reads happening)
- I/O patterns are fairly 'bursty' – lots of time between writes

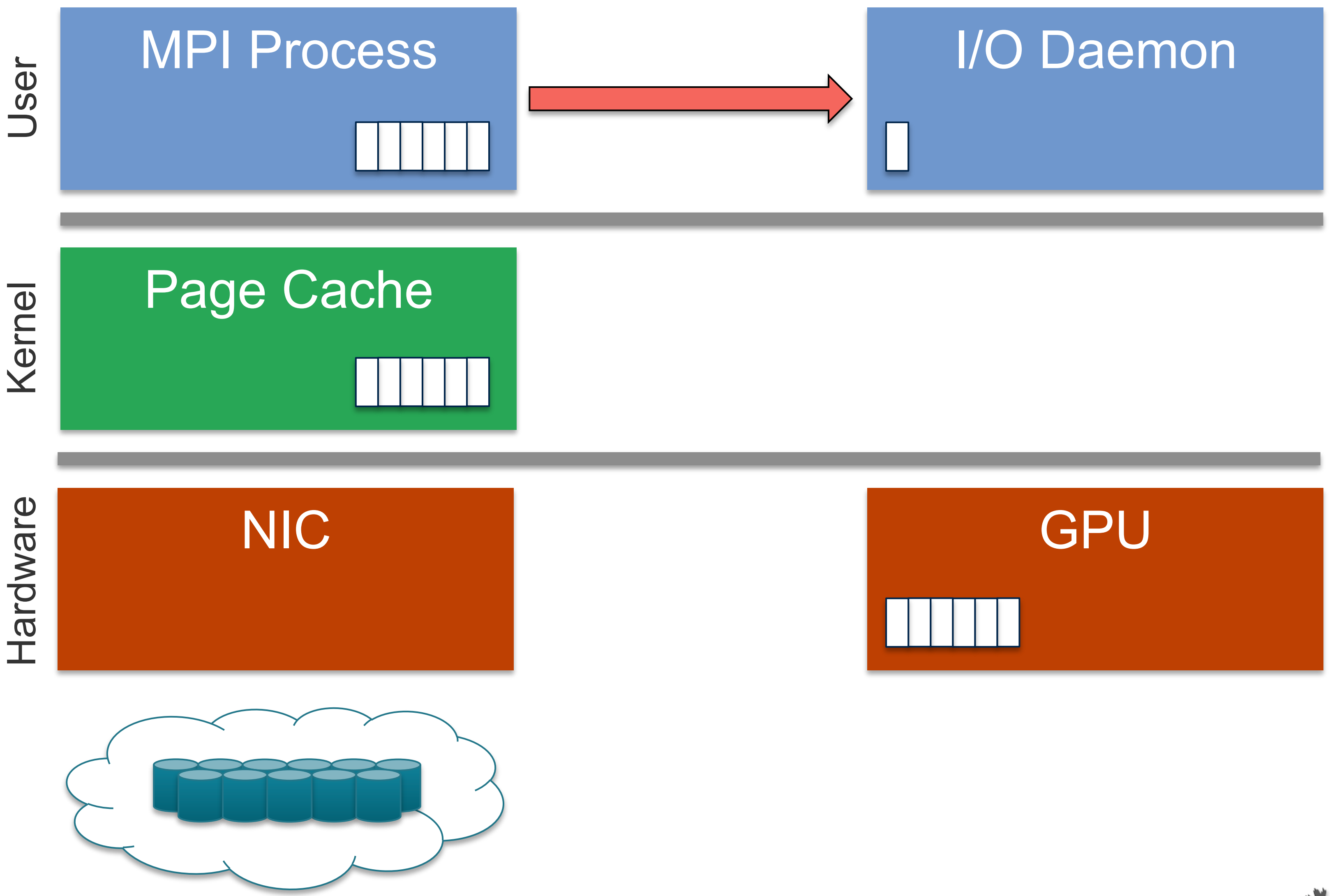
GPU Usage < 100 %

- Measuring delivered compute-hours, GPU usage was around 50% in 2014
- Why aren't all apps using the GPU?
  - Some good reasons, some not-so-good reasons
- "Why" isn't particularly important. What's important is that there's some unused hardware on the nodes. Maybe we can do something interesting/useful with it.
- Let's use the GPU memory to cache filesystem writes

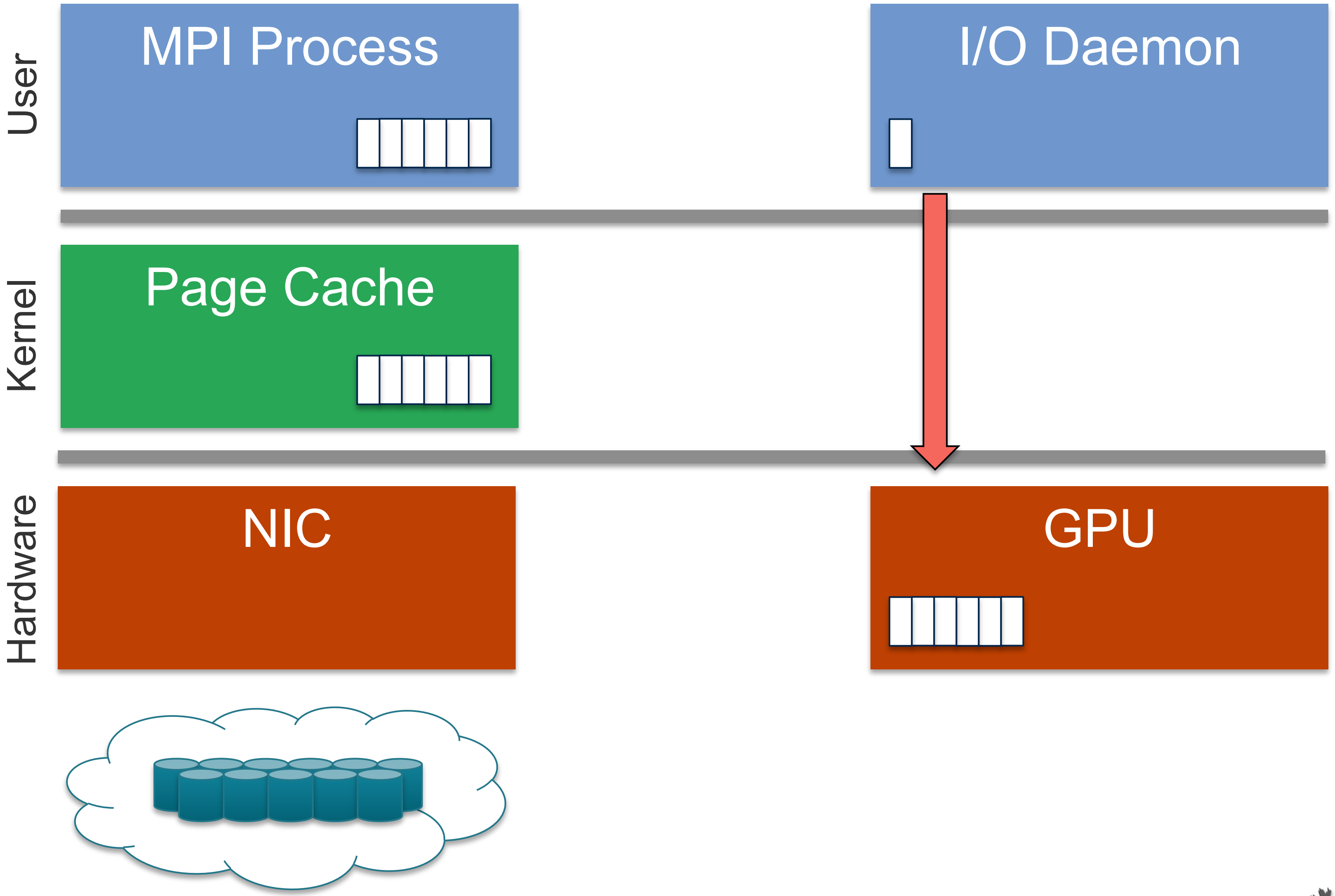
How does all this work?



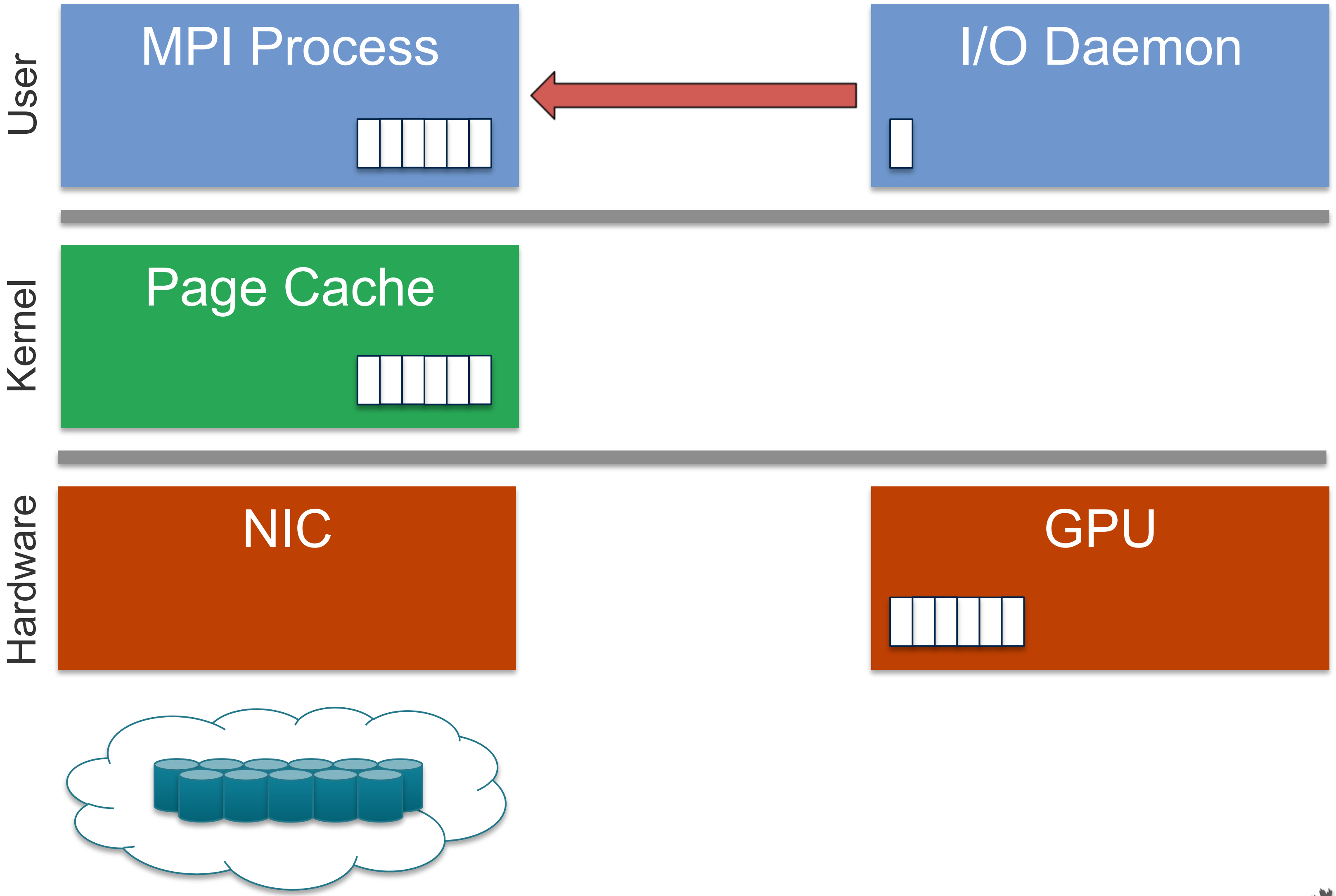
1. Send Write Request



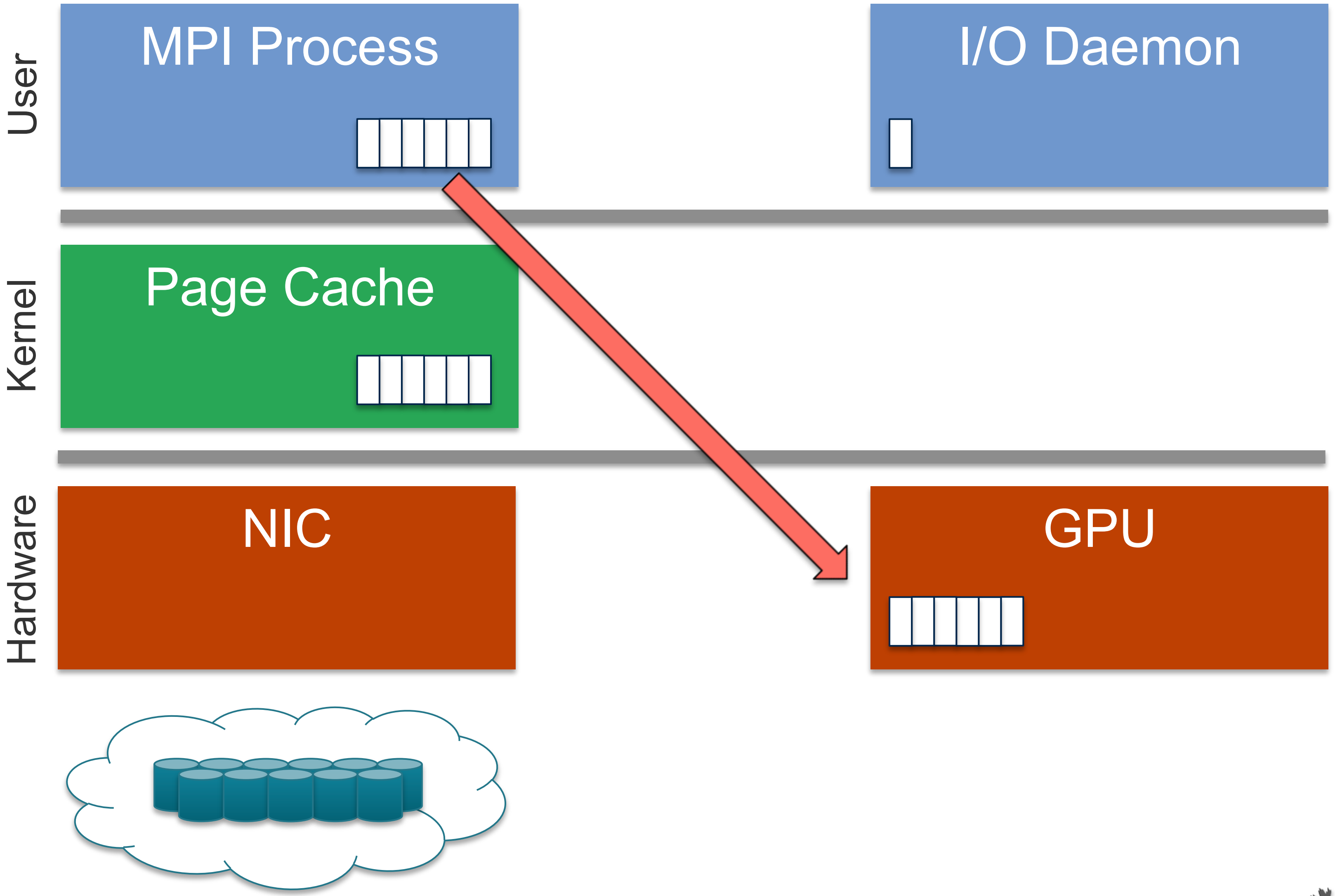
2. Allocate GPU Memory, convert pointer to handle



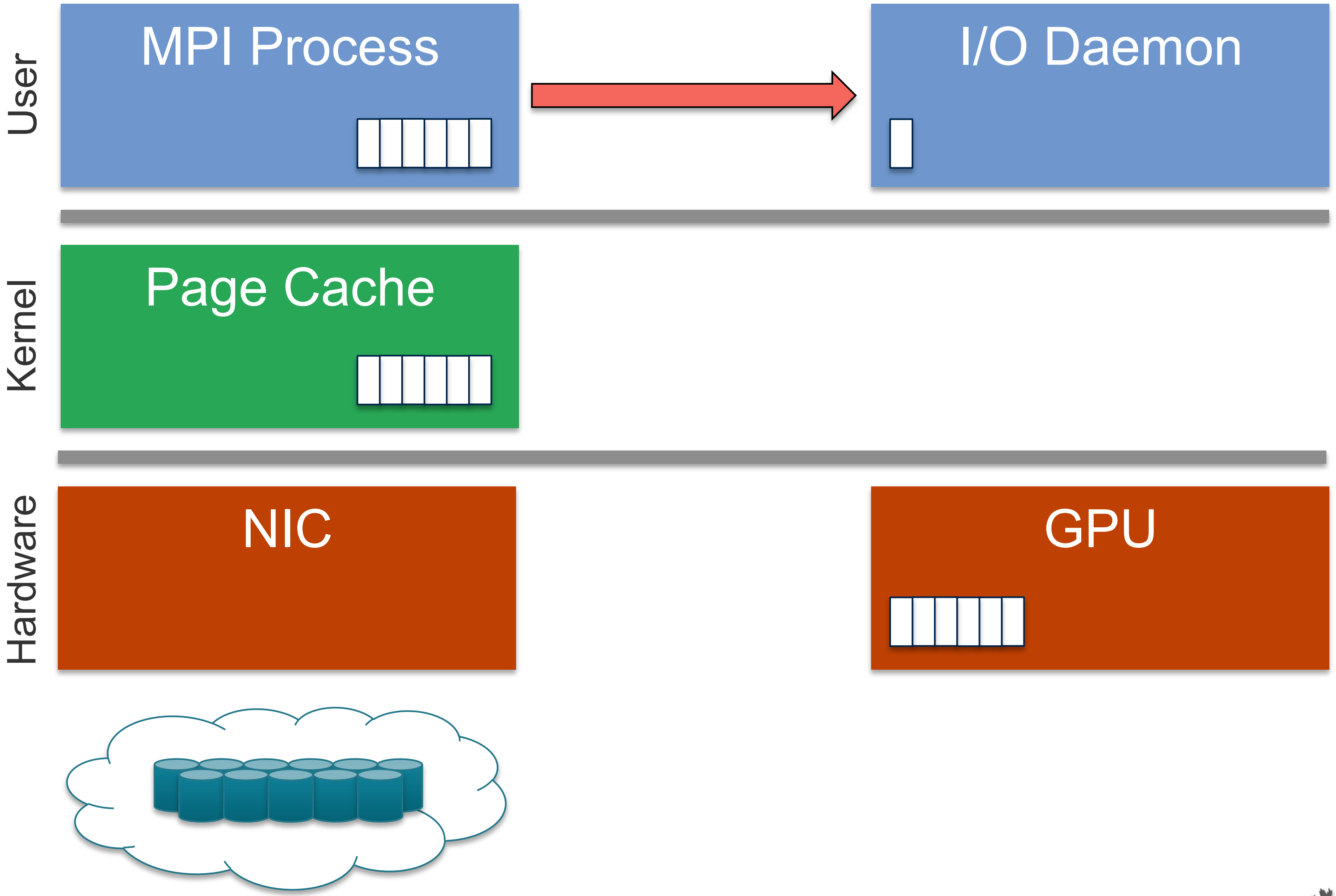
3. Send Reply with Handle



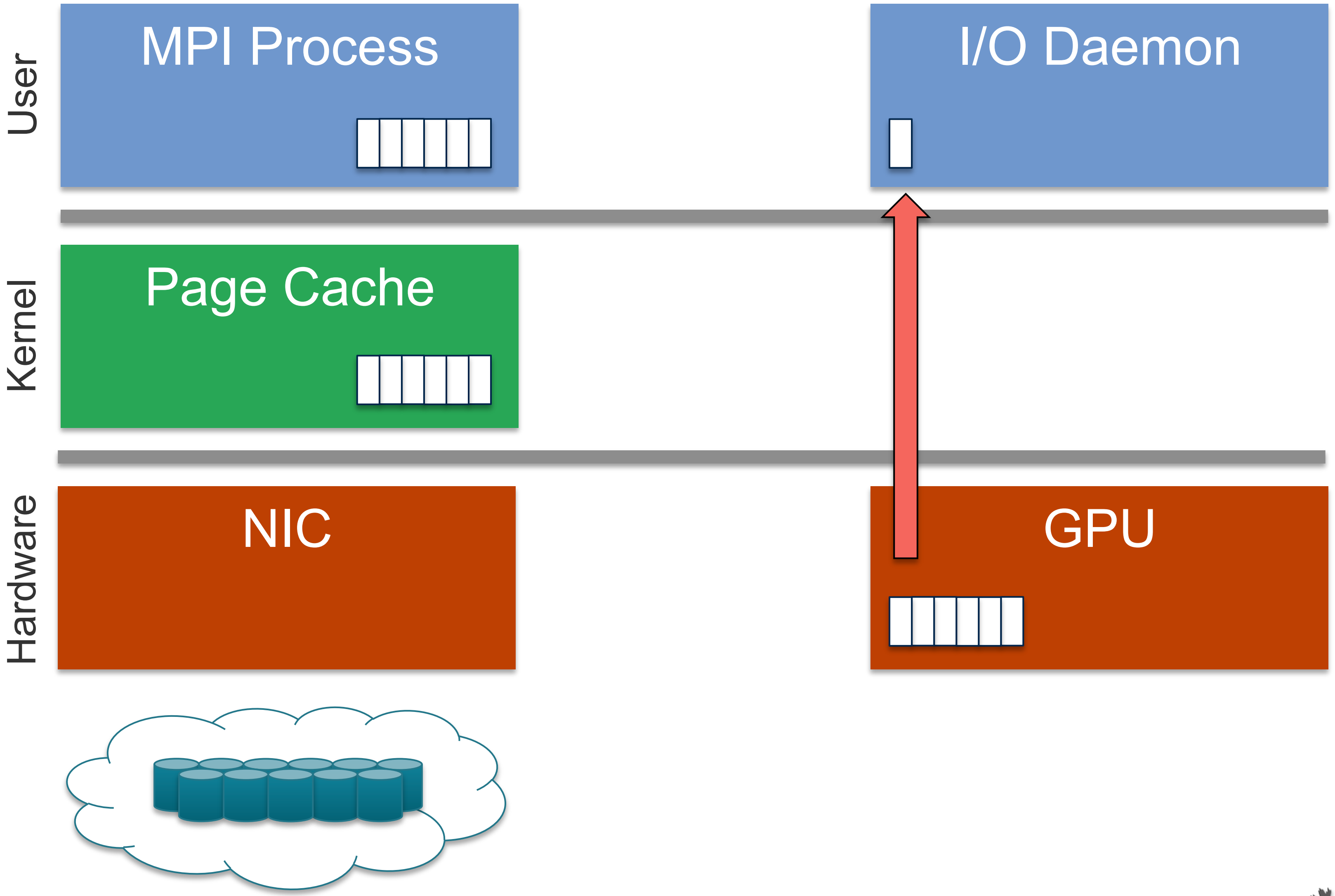
4. Write to GPU Memory



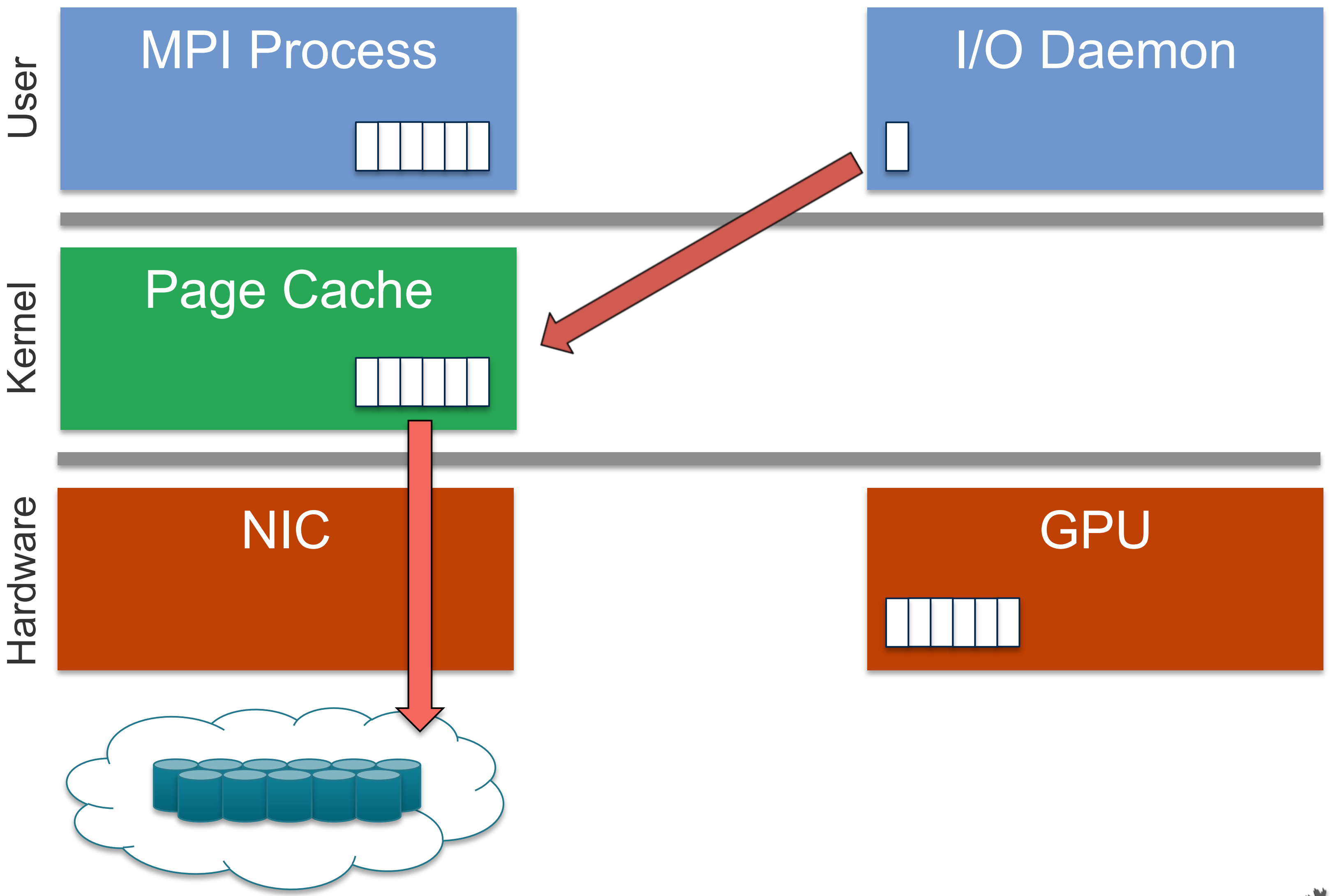
5. Send Write Ready



6. Read from GPU Memory to Daemon Memory



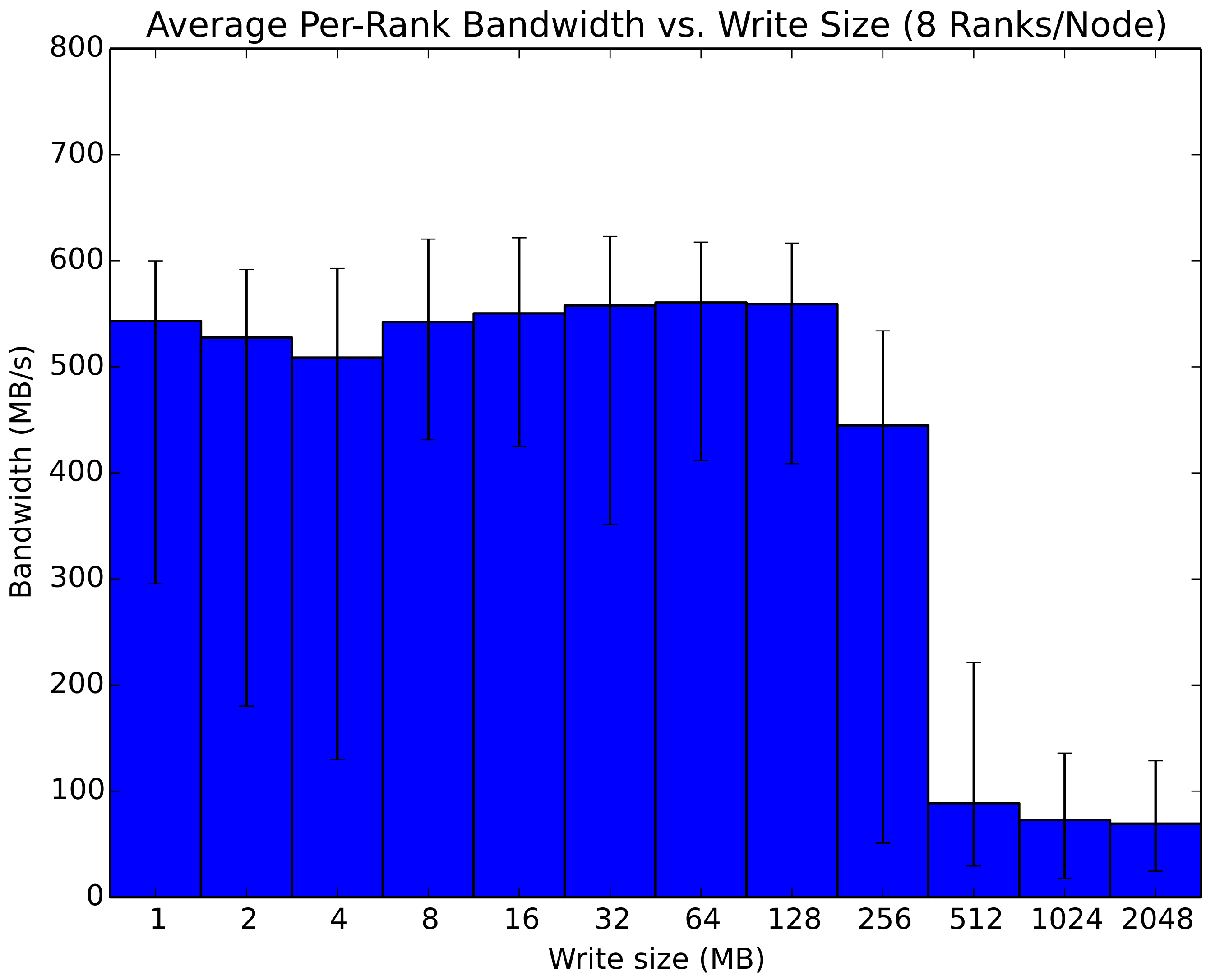
7. Write to Page Cache/PFS



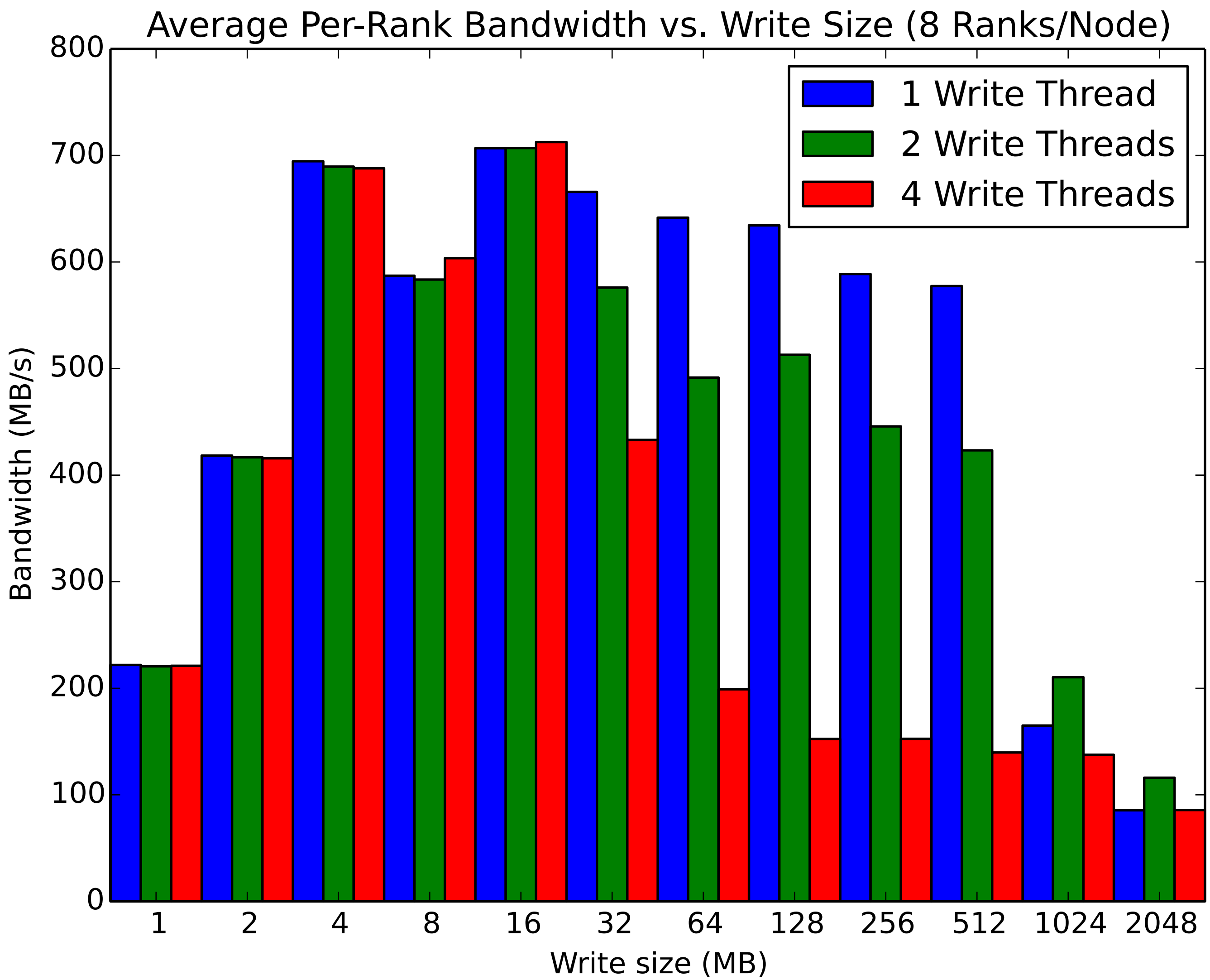
Basic Performance Statistics

- Writing to the filesystem
  - 8 ranks/node, each rank writes to separate file
  - Nothing fancy – just calling write()
  - 550 MB/sec/rank to cache - 100 MB/sec/rank to the filesystem
  - Max write size that fit in cache: 256MB / rank
- Writing to the GPU memory
  - NVidia's bandwidth util says compute nodes can write about 5.5GB/s into GPU memory
  - Our observed aggregate BW was somewhat less, but still much better than writing to the filesystem
  - 550 – 650 MB/sec/rank up to 512MB size

Basic Performance Statistics - Filesystem



Basic Performance Statistics – GPU Mem



Is it worth the effort?

- Much faster than writing straight to the filesystem
- It appears to be a little faster than writing to the Lustre client-side cache
  - Lustre client-side cache needs system memory, which might not be available
- Performance variability should be decreased
  - This is conjecture – trying to get variability numbers is tricky, and it's questionable whether numbers obtained from a synthetic benchmark would be useable anyway.
- Similar improvements with 16 ranks/node
  - Cores are oversubscribed, though

Caveats, Potential Pitfalls

- Data hasn't made it to permanent storage
  - Don't immediate delete your last checkpoint file
- Write only
  - Reads will return what's in the file, not what's in GPU memory
  - No way to verify if a particular write has made it out to the filesystem
- Applications running 16 ranks/node would have to oversubscribe cores to run the daemon
  - For some applications, this might still be a net improvement

Next Steps

- Looking at ways to make this available in a production environment
- We want something that will require minimal modifications to existing code.
- Looking writing a library that will replace existing POSIX calls (open(), write(), etc…) with out own versions
  - Similar to how the MercuryPosix project works
- Also considering modifying existing I/O libraries such as NetCDF.
  - Maintaining the modified libraries might be too much work, though

Conclusions

- Don’t use this technique – port your code
  - Far better to use the GPU hardware for what it was designed: calculations
- If and *ONLY IF* you can’t port you your code, then this technique offers some benefits
- Don’t immediately delete your checkpoint file

Questions?

Acknowledgment: This work was supported by the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is managed by UT Battelle, LLC for the U.S. DOE (under the contract No. DE-AC05-00OR22725).